Practical Scripting

Nathan Dunn
ndunn@cas.uoregon.edu

Craig Rasmussen
rasmus@cas.uoregon.edu

Kristi Potter
kpotter@cas.uoregon.edu
Overview

• Bash Scripting
• Running
• Programming basics
• Python Scripting
• Matlab Scripting
• Source Control
• Vim
Bash Scripting

- Why?
  - Chain command-line tools together
  - Needed for ACISS
- Running
- Fundamentals
- Examples
Running Bash Scripts

• 2 methods for running scripts:
  • Make script executable
  • Use shell to run script
• Run scripts in the background
Running Script

- /bin/sh create_files.sh

NathanDunn@unix-biologists% ./create_files.sh
running script

- ./create_files.sh

NathanDunn@unix-biologists% /bin/sh create_files.sh
running script
Make Executable

- vim create_files.sh

```
#!/bin/sh
echo "running script"
```

Tells system to run with this Make file executable

NathanDunn:unix-biologists% ls -l create_files.sh
-rw-r--r--  1 NathanDunn  staff  36 May 14 15:42 create_files.sh

NathanDunn:unix-biologists% chmod +x create_files.sh
NathanDunn:unix-biologists% ls -l create_files.sh
-rwxr-xr-x  1 NathanDunn  staff  36 May 14 15:42 create_files.sh
Running in Background

- ./create_files.sh &

```
$ cat - &
```

```
$ jobs
```

```
$ fg %1
```

```
$ kill %1
```

- nohup create_files.sh
  - can kill terminal and still active
  - output goes to nohup.out
Bash Fundamentals

- Echo: echo “bob”
- # a comment
- Variables
- Loops
- Conditionals
- Command Arguments
Bash Variable

• Assign a variable:
  • V1='test-v'

• Reference a variable:
  • echo $V1

• Command output to variable ``:  
  • V2 = `pwd`
  • echo $V2

• Append variable:
  • V2 = $V2 + `ls`
Bash Variable

• Append variable:
  • V2 = $V2 + `ls`

• Export variable:
  • export V2  # available external of script

• Unset variable:
  • unset V2
Bash Conditional

• Conditional: if / then / else

    if [ <condition>]; then
      // stuff
    else
      // other stuff
    fi
Bash Conditional

• Conditional: if / then / else
  DIRECTORY=`pwd`/test
  if [ -d "$DIRECTORY" ]; then
    echo “Test directory exists $DIRECTORY”
  else
    echo “$DIRECTORY does not exist”
  fi
Bash For Loop

• “loops”: for loop, while loop

  ```bash
  for i in <array>
  do
    // stuff here with $i
  done
  ```

  ```bash
  while <condition>
  do
    // stuff here
  done
  ```
Bash For Loop

• “loops”: for loop, while loop

```bash
OUTPUT_FILE='abcd.txt'
touch $OUTPUT_FILE
for i in `ls`
do
echo "directory contents $i"
echo $i >> $OUTPUT_FILE
done
```
Command Arguments

- $1 (first)
- $2 (second), etc. etc.
- ./command.sh abcd

```bash
PREFIX=$1
for i in {1..5}
do
touch "$PREFIX_$i.txt"
done
```
Bash Examples

- Sorting Fastq
- Process Fastq Barcodes
- GEO Accessions
- Pipe Separated Data
- Concatenate File Sets
- Remove Duplicate Patterns
Fastq File

- @Header
- Sequence
- +Comment
- Quality

@HWI-ST0747:210:D05GJACXX:3:2308:19482:126261 1:Y:0: CAACTTCCGCGCCCCCAGCGCCCGGAACGCGTACTCCTGTA + =1:;DB:D@)0 ) )2 : : ?3CD: ) ?0 : : ; @A<5−9?D@( ; @
**Sorting FastQ Files**

- combine all headers by group
- header (sample 1) `@IRIS:7:1:17:394#0/1` >> `17.fq`
- header (sample 2) `@IRIS:7:1:17:394#0/2` >> `17.fq`
for line in `cat small_sample_*.fq`
do
  if [[ "$line" == @* ]]
    then
      GROUP=`echo $line | cut -d: -f4 -s`
      echo $GROUP
    fi
  echo $line >> output/$GROUP.fq
done
Process Fastq Barcodes

- Dump all of barcode CAACT into my.fastq
- `grep -P "\@.*\n^CAACT.*\n.*\n" illumina.fastq > my.fastq`
- `sed -e 's/^CAACT//@' my.fastq > processed.fastq`
- remove leading barcode
GEO Accessions


gunzip GPL4014_family.soft.gz

grep -v ^! GPL4014_family.soft | grep -v \# | grep ENSDARG00000 | cut -f2 | sort -u | grep "^B|^NM" > processed_GPL.txt ;
Pipe Separated Data

Gene Expression

ext2|Prim-25|Prim-25|pectoral fin musculature|xpat superterm|
extl3|Long-pec|Long-pec|pectoral fin musculature|xpat superterm|
gli3|High-pec|High-pec|pectoral fin musculature|xpat superterm|

Phenotype

ext2|Prim-5|Prim-15|pectoral fin bud|hypoplastic|abnormal|pheno entity 1 superte|
ext2|Prim-25|Prim-25|apical ectodermal ridge pectoral fin bud|aplastic|abnormal|pheno entity 1 superte|
ext2|Day 4|Day 4|pectoral girdle|decreased size|abnormal|pheno entity 1 superte|
ext2|Protruding-mouth|Protruding-mouth|pectoral fin|decreased length|abnormal|pheno entity 1 superte|

- sort by anatomy
- sort -t'|' -k4 phenoGeneFin.txt > output.txt
Concatenate File Sets

- Combine contents of files with same name in 2 directories

```bash
for i in `ls input1/* | cut -f2 -d\/` 
do
cat input1/$i input2/$i > output_htseq/$i
done
```
Remove Duplicate Patterns

- Duplicate entries, but order is reversed

```bash
ENSDARG000000017007, nr4a2a, 9, 5047097, 5051396, ENSDARG00000044532, nr4a2b, 6, 12373999, 12378404
ENSDARG000000440177, rgs, 6, 34300760, 34303440, ENSDARG00000017860, rgs5b, 2, 6590676, 6593943
ENSDARG00000044532, nr4a2b, 6, 12373999, 12378404, ENSDARG00000017007, nr4a2a, 9, 5047097, 5051396
```

cut -d, -f1-5 Duplicates.csv | cat -n | sort -k2 > first_column.csv
cut -d, -f6-10 Duplicates.csv | cat -n | sort -k2 > second_column.csv
# put them together again
cat first_column.csv second_column.csv > combined_columns.csv

# sort by gene name
# do a unique by gene name to get rid of duplicate genes
# sort by first number
sort -k2 combined_columns.csv > sorted_genes.csv
uniq -s7 sorted_genes.csv | sort > unique_genes.csv
Remove Duplicate Patterns

• Correct, but should be only two lines

1 ENSDARG00000017007,nr4a2a,9,5047097,5051396
1 ENSDARG00000044532,nr4a2b,6,12373999,12378404
2 ENSDARG00000017860,rgs5b,2,6590676,6593943
2 ENSDARG00000040177,rgs16,6,34300760,34303440

# put back together (either order) based on the number
# remove leading numbers
COUNTER='a'
OUTPUT="removed_duplicates.csv"
rm -f $OUTPUT
cut -c8-1000 unique_genes.csv | while read i
do
if [ "$COUNTER" == a ]; then
    COUNTER='b'
    LINE="i"
else
    LINE="$LINE,$i"
    COUNTER='a'
    echo $LINE >> $OUTPUT
fi
done

```
vim
%s/\^.*\t\(.*\)\n^.*\t\(.*\n\)\1,\2/
```
Python

- Object-oriented scripting language
- Easy, readable, lots of doc
- Whitespace!
Bash

• **loop**
  ```bash
  for i in <list>
  do
  <statement>
  done
  ```

• **conditional**
  ```bash
  if [ "$1" == "sometext" ]; then
  <statement>
  else
  <statement>
  fi
  ```

• **read in**
  ```bash
  a=`cat filename1.txt`
  ```

• **write out**
  ```bash
  cat $a > filename2.txt
  ```
Python

• **loop**
  
  ```python
  for i in <list>:
    <statement>
  ```

• **conditional**
  
  ```python
  if var1==var2:
    <statement>
  ```

• **read in**
  
  ```python
  pwd = os.path.abspath(os.path.dirname(__file__))
  input_file = open(pwd+''/inputfile.txt','','r')
  # read input_file
  input_file.close()
  ```

• **write out**
  
  ```python
  output_file = open(pwd+''/outputfile.txt','','w')
  output_file.write(stringsofstuff)
  output_file.close()
  ```
Barcode Reader Python

• For each barcode X dump all fastq sequences to a file named X.fastq

```
@HWI-ST0747:210:D05GJACXX:3:2308:19482:126261 1:Y:0:
CAACT TCCGCGCCCCCAGCGGAAACGCGTACTCCTGTA
+
=1:;DB:D@)0))2::?3CD:)?0::;@A<5–9?D@>(;@

CAACT.fastq
```

```
@HWI-ST0747:210:D05GJACXX:3:2308:19482:126261 1:Y:0:
TCCGCGCCCCCAGCGGAAACGCGTACTCCTGTA
+
B:D@)0))2::?3CD:)?0::;@A<5–9?D@>(;@
```
from Fastq import *

class Fastq:
    def __init__(self):
        self.comment = ""
        self.sequence = ""
        self.barcode = ""
        self.comment2 = ""
        self.score = ""

    def isEmpty(self):
        return self.comment=="" and self.sequence=="" and self.comment2=="" and self.score==""

Barcode Reader Python
Barcode Reader Python

- Use a “dictionary” or “map"
- `<barcode>,<list of sequences>`
- if not found add, otherwise append

```python
barcode_file = {}

def handleBarcode(obj,mydict):
    retrieved_obj = barcode_file.get(fastq_object.barcode)
    if retrieved_obj == None:
        barcode_file[fastq_object.barcode] = [fastq_object]
    else:
        retrieved_obj.append(fastq_object)
    barcode_file[fastq_object.barcode] = retrieved_obj
```
Barcode Reader Python

for line in input_file:
    if count==0:
        if False==fastq_object.isEmpty():
            handleBarcode(fastq_object,barcode_file)
        fastq_object = Fastq()
        fastq_object.comment = line
        count = 1
    elif count==1:
        fastq_object.sequence = line[5:]
        fastq_object.barcode = line[:5]
        count = 2
    elif count==2:
        fastq_object.comment2 = line
        count = 3
    elif count==3:
        fastq_object.score = line
        count = 0
from Bio import SeqIO

original_reads = SeqIO.parse("illumina.fastq", "fastq")
for read in original_reads:
    primer = str(read.seq[0:primer_length])
    trimmed_read = read[primer_length:]
    handleBarcode(trimmed_read, primer, barcode_file)

for i in barcode_file.iterkeys():
    output_file = open(pwd+'output/' + str(i) + '.fastq', 'w')
    for sequence in barcode_file.get(i):
        output_file.write(sequence.format("fastq"))
Install BioPython

- Install:
  - [http://biopython.org/DIST/docs/tutorial/Tutorial.html](http://biopython.org/DIST/docs/tutorial/Tutorial.html)
  - python setup.py build
  - sudo python setup.py install
Matlab

• Entire ecosystem
• Distributed toolkit
• Easy, readable, lots of doc and users
• University License: https://it.uoregon.edu/software/matlab
  • Login at bottom and follow instructions
• On ACISS
Matlab on ACISS

• Run / edit locally

• Send to ACISS (scp or mercurial)
  
  • ssh -X ndunn@aciss.uoregon.edu

• qsub -Xlq short

• module load matlab
Matlab on ACISS

• Interactive:
  • `matlab -nojvm # if no graphing`
  • `matlab -nodesktop -nosplash # if graph`
Matlab on ACISS

- Run in script:
  - `matlab -jvm -r <command>`
  - e.g., run `matlab -jvm -r testfile`
  - runs `testfile.m`
Matlab specifics

• “;” suppresses output
• ‘echo’
• Assign a variable:  a=3 ;
• Matrix:  a=[1, 2, 3; 4, 5, 6];
  • a(1:2,2)
• Reference a variable:
  • echo a
• Specify range:
  • b=1:3
Python

• loop
  for i in <list>:
    <statement>

• conditional
  if var1==var2:
    <statement>

• read in
  
  pwd = os.path.abspath(os.path.dirname(__file__))
  input_file = open(pwd+'/inputfile.txt','r')
  # read input_file
  input_file.close()

• write out
  
  output_file = open(pwd+'/outputfile.txt','w')
  output_file.write(stringsofstuff)
  output_file.close()
Matlab

• **loop**
  
  ```matlab
  for i = <list>
    <statement>
  end
  ```

• **conditional**
  
  ```matlab
  if var1==var2
    <statement>
  end
  ```

• **read in**
  
  ```matlab
  dups = open ('Duplicates.csv')
  dups.textdata(1,1:3)
  ```

• **write out**
  
  ```matlab
  dlmwrite ('data2.txt',d3)
  xlswrite('data2.xls',d2)
  save d2
  ```
Mercurial Source Control

- Way to track file changes
- A way to share file changes
- A way to backup file changes
- A merge of git / svn
Create a Repository

- module load mercurial # on aciss
- Create your own:
  - mkdir myproject1
  - cd myproject1
  - hg init # in directory
  - # edit some files in that directory
  - hg add <files>
  - hg commit
Use a Remote Repository

• Working with another repository:
  • `hg clone`  # how to do a checkout
  • `hg clone ssh://<user>@aciss.uoregon.edu/<repos> <target_directory>`
  • `hg clone <path to directory> <target_directory>`
  • `hg log`  # see what changes
  • `hg diff`
  • `hg push`  # push changes to remote repos
  • `hg pull`  # pull changes from remote repos
Example Code Access

- On ACISS:
  - `hg clone /home2/ndunn/hg/unix_geo_week3`

- Remote:
  - `hg clone ssh://<user>@aciss.uoregon.edu:hg/unix_geo_week3`

- Bitbucket
Edit Scripts

• Set cap-locks to controls!!
• Options:
  • Nano / pico: good to begin with
  • Vim
  • Emacs
• Source Control
Nano

- `nano <filename>`
- `nano small_sample_1.fq`
Vim

- `vim <filename>`
- harder to learn, but very efficient
- You have 3 modes:
  - Command mode: all keystrokes are interpreted as commands (similar to less)
  - Insert mode: most keystrokes are inserted as text (same as nano)
  - Visual mode: helps to visually select some text
Vim Command Mode

- Command Mode:
  - movement is same as less
  - ‘b’ forward, ‘w’ back
  - :w (write)
  - :q (quit)
  - :b (buffer)
  - :e (edit)
  - :%s/<pattern>/<replace>/g
Vim Command Mode

- Copy / paste:
  - yy = copy line
  - p = paste
  - dd = delete line
  - x = delete char
Vim Command Mode

- Undo / Redo
  - u = undo
  - cntrl-r = redo
Vim Command Mode

- Command mode -> Insert mode:
  - i = insert
  - shift-i = insert before line
  - a = insert after character
  - shift-a = insert after line
  - escape = return to Command mode
Vim Visual Mode

• Command mode -> Visual mode:
  • v = visually select spot
  • shift-v = visually select line
  • navigate as in command-mode
Other Resources

• Do it!
• Google it!
• Ask!
• CIS 122 (Python), CIS 399 (All of the above)
Next Time

- Python scientific libraries (and virtual environments).